A Complete Algorithm for a Moving Target Traveling Salesman Problem with Obstacles

Anoop Bhat[1], Geordan Gutow[1], Bhaskar Vundurthy[1], Zhongqiang Ren[2], Sivakumar Rathinam[3], and Howie Choset[1]

[1] Carnegie Mellon University, Pittsburgh PA 15213, USA
[2] Shanghai Jiao Tong University, Shanghai, China
[3] Texas A&M University, College Station, TX 77843

Abstract. *The moving target traveling salesman problem with obstacles* (MT-TSP-O) is a generalization of the traveling salesman problem (TSP) where, as its name suggests, the targets are moving. A solution to the MT-TSP-O is a *trajectory* that visits each moving target during a certain time window(s), and this trajectory avoids stationary obstacles. We assume each target moves at a constant velocity during each of its time windows. The agent has a speed limit, and this speed limit is no smaller than any target's speed. This paper presents the first complete algorithm for finding feasible solutions to the MT-TSP-O. Our algorithm builds a tree where the nodes are agent trajectories intercepting a unique sequence of targets within a unique sequence of time windows. We generate each of a parent node's children by extending the parent's trajectory to intercept one additional target, each child corresponding to a different choice of target and time window. This extension consists of planning a trajectory from the parent trajectory's final point in space-time to a moving target. To solve this point-to-moving-target subproblem, we define a novel generalization of a visibility graph called a moving target visibility graph (MTVG). Our overall algorithm is called MTVG-TSP. To validate MTVG-TSP, we test it on 570 instances with up to 30 targets. We implement a baseline method that samples trajectories of targets into points, based on prior work on special cases of the MT-TSP-O. MTVG-TSP finds feasible solutions in all cases where the baseline does, and when the sum of the targets' time window lengths enters a critical range, MTVG-TSP finds a feasible solution with up to 38 times less computation time.

Keywords: Motion Planning · Traveling Salesman Problem · Combinatorial Search

1 Introduction

Given a set of targets and the travel costs between every pair of targets, the traveling salesman problem (TSP) seeks an order of targets for an agent to visit that minimizes the agent's total travel cost. In the moving target traveling salesman problem (MT-TSP) [10], the targets are moving through free space, and we seek not only an order of targets, but a trajectory for the agent intercepting

each target. The agent's trajectory is subject to a speed limit and must intercept each target within a set of target-specific time intervals, called time windows. We consider the case where travel cost between targets is equal to the travel time: this cost is not fixed *a priori*, as in the TSP, and instead depends on the time at which the agent intercepts each target. Prior work on the MT-TSP assumes that the agent's speed limit is no smaller than the speed of any target [10,18,19,26], and we make the same assumption in our work. When there are moving targets as well as obstacles for the agent to avoid, we refer to the problem as the moving target traveling salesman problem with obstacles (MT-TSP-O), shown in Fig. 1.

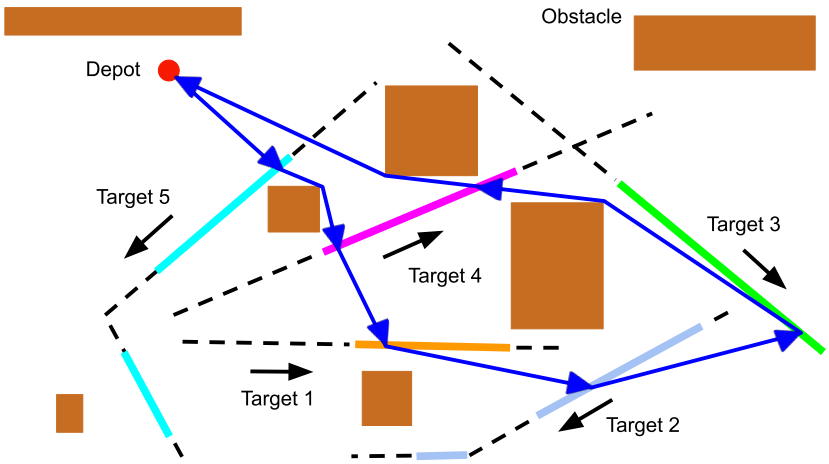


Fig. 1. Targets move along trajectories with piecewise-constant velocities, which can be intercepted by agent during time windows depicted in bold colored lines. Agent's trajectory shown in dark blue avoids obstacles, intercepts each target within its time window, and returns to start location (depot).

Two properties we desire for an MT-TSP-O algorithm are completeness[4] and optimality. No algorithm for the MT-TSP-O in the literature has either of these properties. Guaranteeing completeness is complicated by the fact that even the problem of finding a feasible solution is NP-complete, since the MT-TSP-O generalizes the TSP with time windows (TSP-TW) [21]. In this paper, we present the first complete algorithm for the MT-TSP-O.

Simpler cases of the MT-TSP-O have been addressed in the literature, with completeness guarantees in some cases. For example, in the absence of obstacles, [20] provides a complete and optimal solver for the MT-TSP assuming targets move at constant velocities. [22] provides a complete and optimal method when targets have piecewise-constant velocities. Heuristics are presented in [1,3,7,8,11, 17,18,25,26] for variants of the MT-TSP, only guaranteeing completeness in the

[4] Completeness refers to an algorithm's guarantee on finding a feasible solution when a problem instance is feasible or reporting infeasible in finite time otherwise [2,13].

absence of time windows. In the presence of obstacles, there is one related work [14] that considers the case where the agent is restricted to travel along a straight-line path when moving from one target to the next, providing an incomplete algorithm. A generic approach to solving these special cases of the MT-TSP-O is to sample the trajectories of targets into points, find agent trajectories between every pair of points, then select a sequence of points to visit by solving a generalized traveling salesman problem (GTSP) [14, 19, 23]. This approach is not complete, since it may only be possible for the agent to intercept some target at a time that is in between two of the sampled points in time representing the target's trajectory.

We develop a new algorithm for the MT-TSP-O that guarantees completeness. Our algorithm, called MTVG-TSP, leverages a novel generalization of a visibility graph, which we call a moving target visibility graph (MTVG), which enables us to plan a trajectory from a starting point in space-time to a moving target. Given a sequence of time windows of targets, we can find a minimum-time agent trajectory intercepting each target within its specified time window via a sequence of A* searches, each on a MTVG. In particular, we can do so without sampling any target's trajectory, avoiding the limitations of prior work. By performing a higher level search for a sequence of time windows and computing an agent trajectory for each generated sequence, MTVG-TSP finds an MT-TSP-O solution if one exists. We extensively test our algorithm on problem instances with up to 30 targets, varying the length and number of time windows, and we compare our algorithm's computation time to a method based on prior work [14, 19, 23] that samples trajectories of targets into points. We demonstrate that when the sum of the time window lengths for each target enters a critical range, the sampled-points method requires an excessive number of sample points to find a feasible solution, while our method finds solutions relatively quickly.

2 Problem Setup

We consider a single agent and N_τ targets, all moving on a 2D plane ($\mathbb{R}^2$). The trajectory of target $i \in [N_\tau]$[5] is denoted as $\tau_i : \mathbb{R}^+ \to \mathbb{R}^2$. Each target has a set of N_i time windows $\{w_{i,1}, w_{i,2}, \ldots, w_{i,N_i}\}$, where $w_{i,j} = [t^0_{i,j}, t^f_{i,j}]$ is the j^{th} time window of target i. Target i moves at a constant velocity within each of its time windows, but its velocity may be different for each time window. Given a final time T^f, denote the trajectory for the agent as $\tau_A : [0, T^f] \to \mathbb{R}^2$. τ_A must start and end at a given point called the depot denoted as d with position $p_d \in \mathbb{R}^2$. The agent can move in any direction with a speed at most v_{max}.

Let $\{O_1, O_2, \ldots O_{N_O}\}$ denote the set of obstacles where N_O denotes the number of obstacles. We define $\Psi(t^0, t^f)$ as the set of all the feasible agent trajectories defined on the time interval $[t^0, t^f]$ such that for any time $t \in [t^0, t^f]$, the agent satisfies the speed constraint and its position never enters the interior of any obstacle. We assume that within target i's time windows, target i does not move with speed greater than v_{max} and does not enter the interior of any obstacle.

[5] For a positive integer x, $[x]$ denotes the set $\{1, 2, \cdots, x\}$.

We say that a trajectory τ_A for the agent *intercepts* target $i \in [N_\tau]$ if there exists a time t such that for some $j \in [N_i]$, $t \in w_{i,j}$ and $\tau_A(t) = \tau_i(t)$. We define the MT-TSP-O as the problem of finding a final time T^f and an agent trajectory $\tau_A \in \Psi(0, T^f)$ such that τ_A starts and ends at the depot, intercepts each target $i \in [N_\tau]$, and T^f is minimized.

3 MTVG-TSP Algorithm

The MTVG-TSP algorithm interleaves a higher-level search on a *time window graph* and a lower-level search on a *moving target visibility graph* (MTVG). The nodes in the time window graph, called *window-nodes*, each represent either the depot or a pairing of a target with one of its time windows. A feasible solution to the MT-TSP-O corresponds to a cycle in the time window graph containing the depot and exactly one window-node per target. Our algorithm finds such a cycle by building a *trajectory tree*, where each *tree-node* contains a sequence of window-nodes and an associated agent trajectory. A tree-node's children are each generated in two steps. First, we append a window-node to the end of the tree-node's window-node sequence. Then we extend the tree-node's agent trajectory to intercept the appended window-node's target. We perform this trajectory extension via the MTVG. In particular, we construct the MTVG by augmenting a standard visibility graph with the window-node whose target we aim to intercept. Extending a tree-node's trajectory consists of planning a path in the MTVG from the final point in the tree-node's trajectory to the added window-node. When we have a tree-node with a trajectory that intercepts all targets and returns to the depot, we have a solution to the MT-TSP-O.

3.1 Window-Nodes

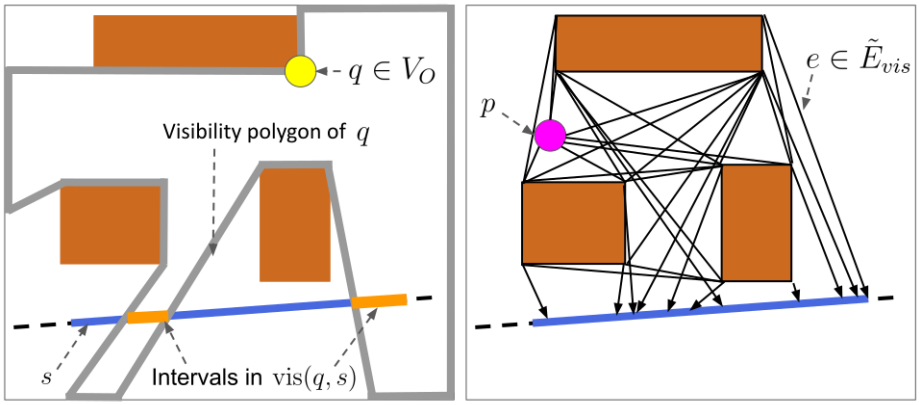
Since we will be constructing two novel graphs, the time window graph and the moving target visibility graph, each containing a common type of node called a *window-node*, we define window-nodes here. A window-node $s = (i, t^0_{i,j}, t^f_{i,j})$ is an association of a target i with one of its time windows $w_{i,j} = [t^0_{i,j}, t^f_{i,j}]$. The set of all window nodes is $V_{tw} = \{(0, 0, \infty)\} \cup \bigcup_{i \in [N_\tau]} \bigcup_{j \in [N_i]} \{(i, t^0_{i,j}, t^f_{i,j})\}$. In addition to having a window-node for every possible pair of target and time window, we have a window-node $s_d = (0, 0, \infty)$ for the depot. The depot is viewed as a fictitious target 0 with $\tau_0(t) = p_d$ for all t. For each window-node $s = (i, t^0_{i,j}, t^f_{i,j})$, we define the functions $\text{targ}(s) = i$, $t^0(s) = t^0_{i,j}$, and $t^f(s) = t^f_{i,j}$. We say an agent trajectory τ_A *intercepts* $s \in V_{tw}$ at time $t \in [t^0(s), t^f(s)]$ if $\tau_A(t) = \tau_{\text{targ}(s)}(t)$.

3.2 Initial Visibility Computations

Next, we describe two initial data structures needed to construct the moving target visibility graph. The first is a standard visibility graph $G_{vis} = (V_{vis}, E_{vis})$

[16]. Let $V_O \subseteq \mathbb{R}^2$ be the set of convex vertices of all the obstacles[6]. The set of nodes V_{vis} in G_{vis} contains vertices in V_O, the depot position, and the positions of each target's trajectory at the beginning and end of each of its time windows, i.e. $V_{vis} := V_O \cup \{p_d\} \cup \{\tau_{\text{targ}(s)}(t^0(s)) : s \in V_{tw} \setminus \{s_d\}\} \cup \{\tau_{\text{targ}(s)}(t^f(s)) : s \in V_{tw} \setminus \{s_d\}\}$. We draw an edge from $q \in V_{vis}$ to $q' \in V_{vis}$ if q' is contained in the visibility polygon of q, denoted as $\text{vpoly}(q)$ (computed using CGAL [24]).

The second data structure encodes visibility relationships between points in space and window-nodes. In particular, for each $q \in V_{vis}$, $s \in V_{tw}$, we compute a *visible interval set* $\text{vis}(q, s)$, containing every interval $I \subseteq [t^0(s), t^f(s)]$ such that for all $t \in I$, we have $\tau_{\text{targ}(s)}(t) \in \text{vpoly}(q)$. We illustrate $\text{vis}(q, s)$ in Fig. 2 (a). The set of all visible interval sets is $\Lambda_{vis} = \{\text{vis}(q, s) : \forall q \in V_{vis}, s \in V_{tw}\}$.



(a) Constructing $\text{vis}(q, s)$ (b) Constructing $\tilde{G}_{vis}$

Fig. 2. (a) We compute $\text{vis}(q, s)$ for all $q \in V_O$ and $s \in V_{tw}$. (b) In the moving target visibility graph (MTVG) associated with p and s, we draw an edge from q to s if $\text{vis}(q, s) \neq \emptyset$. All edges between position-nodes are bidirectional, but edges from position-nodes to s are unidirectional: there are no edges leaving s. The positions of endpoints of edges on s are drawn arbitrarily, since the agent's position after traversing edge (q, s) depends on the time when the agent leaves q.

3.3 Moving Target Visibility Graph

A recurring subproblem in our algorithm is to find a trajectory from a point (p, T) in space-time that intercepts a particular window-node s in minimum time. For each of these subproblems, we define a graph $\tilde{G}_{vis} = (\tilde{V}_{vis}, \tilde{E}_{vis})$ called a *moving target visibility graph* (MTVG). The set of nodes is $\tilde{V}_{vis} = V_{vis} \cup \{p, s\}$, where nodes in $V_{vis} \cup \{p\}$ are called *position-nodes*, and s is the window-node we aim to intercept. The set of edges is $\tilde{E}_{vis} = E_{vis} \cup E_p \cup E_s$. We construct E_p by

[6] A convex obstacle vertex is is a vertex where the internal angle between the two incident edges is less than π radians. Using only convex vertices in a visibility graph reduces graph size without discarding shortest paths through the graph [15]

drawing edges from p to all $q \in V_{vis}$ such that $q \in \text{vpoly}(p)$; if we already have $p \in V_{vis}$, we can skip this step, since all possible edges from p to $q \in V_{vis}$ already exist in E_{vis}. Constructing E_s consists of two steps. First, we compute $\text{vis}(p, s)$; if $p \in V_{vis}$, we can skip this step, because we computed $\text{vis}(p, s)$ in Section 3.2. Second, we draw an edge to s from any position-node q, including p, such that $\text{vis}(q, s) \neq \emptyset$, as shown in Fig. 2 (b).

The cost of edge $(u, v) \in \tilde{E}_{vis}$ depends on u, v, and a time variable t representing the time at which the agent departs from node u. We denote the cost of (u, v) at time t as $\tilde{c}_{vis}(u, v, t)$. For edges between position-nodes u and v, the time variable is not used: the edge cost is simply equal to the agent's minimum travel time from u to v: $\tilde{c}_{vis}(u, v, t) = \frac{\|u-v\|_2}{v_{max}} \; \forall t$. For any edge $e_s = (q, s) \in E_s$, the time variable is used to compute the cost as follows:

$$\tilde{c}_{vis}(q, s, t) = \min_{I \in \text{vis}(q,s)} SFT(q, t, s, I) \tag{1}$$

where SFT stands for shortest feasible travel, as in [19]. "Shortest" refers to shortest time. The SFT from point (q, t) to window-node s on interval I is the optimal cost of optimization Problem 2:

$$SFT(q, t, s, I) = \min_{t_s \in I} \quad t_s - t \tag{2a}$$

$$\text{s.t.} \quad \frac{\|\tau_{\text{targ}(s)}(t_s) - q\|}{t_s - t} \leq v_{max} \tag{2b}$$

Problem 2 computes the minimum travel time of a straight-line trajectory starting from (q, t) and intercepting s within interval I, and can be solved in closed-form using methods from [19]. The SFT computation does not consider obstacle-avoidance, but in our case, it does not need to: any straight-line trajectory from q to s intercepting s within some $I \in \text{vis}(q, s)$ is already obstacle-free.

After constructing $\tilde{G}_{vis}$, we find a minimum-time trajectory from (p, T) to s via an A* search [9] from p to s on $\tilde{G}_{vis}$, shown in Alg. 1. Since edge costs in $\tilde{G}_{vis}$ encode travel time, the g-value $g(v)$ for a node v is the shortest travel time out of all paths to v that A* has explored so far. $T + g(v)$ is then the earliest known arrival time to node v. In Line 11, we use this arrival time to compute edge costs from v to its successors. The heuristic $h(q)$ for a position-node q is the Euclidean distance from q to the spatial line segment defined by s, divided by v_{max}, underestimating the travel time from q to s. We set $h(s) = 0$. When A* finds a path to s, the ConstructTrajectory function (Line 19) performs a standard backpointer traversal to obtain a path $Q = (p, q^1, \ldots, q^{N-1}, s)$ through $\tilde{G}_{vis}$, moves the agent at max speed between each position-node in Q, and finally executes the straight-line trajectory associated with $\tilde{c}_{vis}(q^{N-1}, s, T + g(q^{N-1}))$. The result is a minimum-time trajectory from (p, T) to s, if one exists. If such a trajectory does not exist, the condition $g_{cand}(v') \leq t^f(s) - T$ on Line 12 for adding v' to OPEN ensures the search terminates, described in Section 4.

Algorithm 1: A* search from initial point (p, T) to window-node s

Function `PointToMovingTargetSearch`$(p, T, s, G_{vis}, \Lambda_{vis})$**:**
  $\tilde{G}_{vis} = (\tilde{V}_{vis}, \tilde{E}_{vis}) =$ `ConstructMTVG` $(p, s, G_{vis}, \Lambda_{vis})$;
  OPEN = [];
  CLOSED = [];
  Insert p into OPEN with $f(p) = 0$;
  Set $g(v) = \infty$ for all $v \in \tilde{V}_{vis}$ with $v \neq p$. Set $g(p) = 0$.
  while OPEN is not empty and $f(s) > \min_{v \in \text{OPEN}} f(v)$ **do**
    Remove v with smallest $f(v)$ from OPEN;
    Insert v into CLOSED;
    for v' in v.successors() **do**
      $g_{cand}(v') = g(v) + \tilde{c}_{vis}(v, v', T + g(v))$;
      if $g_{cand}(v') < g(v')$ and $v' \notin$ CLOSED and $g_{cand}(v') \leq t^f(s) - T$ **then**
        $g(v') = g_{cand}(v')$;
        Insert v' into OPEN with $f(v') = g(v') + h(v')$;
      end
    end
  end
  if $f(s) \neq \infty$ **then**
    return ConstructTrajectory(s), $g(s) + T$, FEASIBLE;
  end
  return NULL, ∞, INFEASIBLE;

3.4 Time Window Graph

The MTVG defined in Section 3.3 enables finding a minimum-time trajectory intercepting a single window-node. We will need to chain these trajectory computations together to intercept a sequence of window-nodes, containing one window-node per target. To determine this sequence, we define a *time window graph* denoted as $G_{tw} = (V_{tw}, E_{tw})$. The set of nodes in G_{tw} is the set of all window nodes V_{tw} from Section 3.1. We add an edge (u, v) to E_{tw} if there exists an agent trajectory that intercepts u at time t_u and v at time t_v with $t_u \leq t_v$, satisfying speed limit and obstacle avoidance constraints. In particular, we search for such a trajectory that departs at the latest feasible departure time from u to v, denoted as $LFDT(u, v)$. Extending the idea from [19], $LFDT(u, v)$ is the optimal cost of optimization Problem 3:

$$LFDT(u, v) = \max_{\substack{t_u \in [t^0(u), t^f(u)], \\ \tau_A}} \quad t_u \tag{3a}$$

$$\text{s.t.} \quad \tau_A \in \Psi(t_u, t^f(v)), \tag{3b}$$

$$\tau_A(t_u) = \tau_{\text{targ}(u)}(t_u), \tag{3c}$$

$$\tau_A(t^f(v)) = \tau_{\text{targ}(v)}(t^f(v)). \tag{3d}$$

Remark 1. Constraint (3d) requires the agent to meet v at its end time $t^f(v)$, raising the question of whether the agent could depart later from u if we relaxed (3d) and only required the agent to meet v at some time $t_v \in [t^0(v), t^f(v)]$. Since we assumed $\tau_{\text{targ}(v)}$ neither exceeds the agent's maximum speed nor enters the interior of an obstacle during $\text{targ}(v)$'s time window, this relaxation would not let the agent depart later from u. If the agent can depart at time t_u and meet $\text{targ}(v)$ at some time $t_v \in [t^0(v), t^f(v)]$, the agent can follow $\tau_{\text{targ}(v)}$ from time t_v to time $t^f(v)$, thereby meeting v at time $t^f(v)$ as well.

Problem 3 seeks a trajectory that starts by intercepting a window-node and terminates at a prescribed point. We transform Problem 3 via a time reversal to instead seek a trajectory that starts at a prescribed point and terminates by intercepting a window-node. We do so because the transformed problem can be solved using Alg. 1. The transformation defines a fictitious node $\underline{u} = (-\text{targ}(u), -t^f(u), -t^0(u))$ associated with fictitious target $\text{targ}(\underline{u}) = -\text{targ}(u)$. The trajectory of $\text{targ}(\underline{u})$ is $\tau_{-\text{targ}(u)}(t) = \tau_{\text{targ}(u)}(-t)$. Consider a reversed time variable $\underline{t} = -t$. An agent trajectory that departs as late as possible from u, measured in conventional time t, arrives at $\underline{u}$ as early as possible, measured in reversed time $\underline{t}$. We find $LFDT(u, v)$ by solving the transformed Problem 4:

$$LFDT(u,v) = -\min_{\substack{\underline{t}_u \in [t^f(\underline{u}), t^0(\underline{u})], \\ \underline{\tau}_A}} \quad \underline{t}_u \tag{4a}$$

$$\text{s.t.} \quad \underline{\tau}_A \in \Psi(-t^f(v), \underline{t}_u), \tag{4b}$$

$$\underline{\tau}_A(t_u) = \tau_{\text{targ}(\underline{u})}(\underline{t}_u), \tag{4c}$$

$$\underline{\tau}_A(-t^f(v)) = \tau_{\text{targ}(v)}(t^f(v)). \tag{4d}$$

We solve Problem 4 using Alg. 1, and if we find a feasible solution, we draw an edge from u to v in G_{tw} and store $LFDT(u, v)$ with that edge.

3.5 Trajectory Tree

Any feasible agent trajectory τ_A for the MT-TSP-O must intercept each target during one of its time windows before returning to the depot. This means that τ_A intercepts a sequence of window-nodes $(s^1, s^2, \ldots, s^{N_\tau})$ containing exactly one window-node per target, implying the existence of the cycle $S = (s_d, s^1, s^2, \ldots, s^{N_\tau}, s_d)$ in G_{tw}. In this paper, we use the depth-first search (DFS) in Alg. 2 to find a feasible trajectory and its corresponding cycle (if one exists). While we apply DFS in this work to quickly find feasible solutions, other search methods can be used as well, e.g. best-first search to find the global optimum.

Alg. 2 maintains a stack that stores tuples (S, τ_A, T), where S is a path (sequence of window-nodes) through G_{tw}, and (τ_A, T) is a partial solution to the MT-TSP-O, in that (τ_A, T) starts at the depot, avoids obstacles, satisfies the speed limit, and intercepts each window-node in S in order. As the search proceeds, we construct a tree of these tuples, which we call a *trajectory tree*. We refer to tuples (S, τ_A, T) as *tree-nodes*. Each loop iteration begins by popping

a tree-node (S, τ_A, T) from the stack (Line 4), then iterating over the successor window-nodes of (S, T) (Line 6), where $s' \in G_{tw}$ is a successor window-node of (S, T) if and only if all of the following conditions hold:

1. $(s, s') \in E_{tw}$ and $T \leq LFDT(s, s')$, s being the terminal window-node of S.
2. $\text{targ}(s') \neq \text{targ}(s)$ for any s in S.
3. If $s' = s_d$, then S contains one window-node per target.

Algorithm 2: Constructing a cycle through G_{tw} and a trajectory intercepting each node in the cycle. See Alg. 1 for the PointToMovingTargetSearch function. See Section 3.5 for details on the Lookahead function.

```
1  Function DFS(S, G_vis, Λ_vis):
2      STACK = [ (s_d, NULL, 0) ];
3      while STACK is not empty do
4          (S, τ_A, T) = STACK.pop();
5          SUCCESSORS = [];
6          for s' in GetSuccessorWindowNodes(S, T) do
7              τ̄'_A, T', status = PointToMovingTargetSearch
                (τ_A(T), T, s', G_vis, Λ_vis);
8              S' = Append (S, s');
9              if Lookahead (S', T') is INFEASIBLE then
10                 continue;
11             end
12             τ'_A = ConcatenateTrajectories (τ_A, τ̄'_A);
13             if s' is s_d then
14                 return τ'_A, T' FEASIBLE;
15             end
16             SUCCESSORS.append((S', τ'_A, T'));
17         end
           // Add successors to stack in order of decreasing
              final time
18         SUCCESSORS.sortLargestToSmallestT();
19         for (S', τ'_A, T') in SUCCESSORS do
20             STACK.push((S', τ'_A, T'));
21         end
22     end
23     return NULL, ∞, INFEASIBLE;
```

For each successor window-node s', we generate a successor tree-node (S', τ'_A, T') by generating a trajectory $\bar{\tau}'_A$ that begins at $(\tau_A(T), T)$ and intercepts s'. We plan $\bar{\tau}'_A$ using Alg. 1, obtaining final time $T' = T + g(s')$. Condition 1 in the definition of a successor window-node ensures that we always find a trajectory in this step. Next, we generate a path S' through G_{tw} (or a cycle through G_{tw}, if $s' = s_d$) by appending s' to S (Alg. 2, Line 8). Then we perform

a check denoted as Lookahead in Line 9. In particular, let $\Gamma_{unvisited}$ be the set of targets unvisited by $S' = (s_d, s^1, s^2, \ldots, s')$:

$$\Gamma_{unvisited} = [N_\tau] \setminus \{\text{targ}(s^i) : i \in [|S'| - 1]\}. \tag{5}$$

where $|S'|$ is the length of S'. If inequality (6) holds for some $i \in \Gamma_{unvisited}$,

$$T' > \max_{s'' \in \{s''' \in V_{tw} : \text{targ}(s''') = i\}} LFDT(s', s''). \tag{6}$$

then the agent has arrived at s' too late to intercept target i in the future, making it impossible to intercept the set of unvisited targets. Thus we do not add S' as a successor to S. This Lookahead check is analogous to Test 1 in [6]. While it is not needed for completeness, it reduces computation time.

If the Lookahead check succeeds, we concatenate $\bar{\tau}'_A$ with τ_A to obtain trajectory τ'_A, and thereby a partial solution (τ'_A, T') corresponding to S' (Line 12. If $s' = s_d$, then (τ'_A, T') is a feasible solution to the MT-TSP-O and we return (Line 14). Otherwise, we add (S', τ'_A, T') to the list of successors of S (Line 16). After obtaining all successors, we push the successors onto the stack in order of decreasing final time (Lines 18-21), so the next popped tree-node will be the successor of (S, τ_A, T) with the earliest final time.

4 Theoretical Analysis

In this section, we state MTVG-TSP's completeness theorems and sketch the proofs, providing full proofs in Appendix C in the supplementary material.

Theorem 1. *Alg. 1 finds a minimum-time trajectory beginning at (p, T) and intercepting s, if one exists.*

Theorem 1's proof is similar to the admissibility proof of A* [9], differing since the cost of an edge (q, s) in the MTVG depends on $g(q)$ in the A* search. In general, such path-dependent edge costs can make it necessary for an optimal path in a graph to take a suboptimal path to some intermediate node. The bulk of Theorem 1's proof lies in showing that this is never necessary in the MTVG.

Definition 1. *Let (τ^*_A, T^*) be a solution to the MT-TSP-O. (τ_A, T) is a* prefix *of (τ^*_A, T^*) if $\tau_A(t) = \tau^*_A(t)$ for all $t \in [0, T]$ and $T \leq T^*$.*

Theorem 2. *If an MT-TSP-O instance is feasible, MTVG-TSP finds a feasible solution.*

Proving Theorem 2 requires showing that Alg. 2 terminates, and that it terminates by returning a feasible MT-TSP-O solution, as opposed to terminating on Line 23 due to an empty stack. Termination is guaranteed because Alg. 2's search tree is finite. We show that termination is not caused by an empty stack via an induction argument proving that there is always a prefix of a feasible MT-TSP-O solution on the stack. We detail the induction step here.

Suppose a tuple (S, τ_A, T) is popped from the stack with (τ_A, T) a prefix of some feasible MT-TSP-O solution (τ_A^*, T^*). Let $p = \tau_A(T)$. Of the window-nodes intercepted by τ_A^* but not τ_A, let s' be the window-node intercepted earliest. Since τ_A^* travels feasibly from (p, T) to s', arriving at some time $T^{*\prime}$, Theorem 1 implies that Alg. 1 generates a trajectory $\bar{\tau}_A'$ from (p, T) to s', arriving with $T' \leq T^{*\prime}$. Under the assumptions that the targets move no faster than the agent's maximum speed and do not enter the interior of obstacles during their time windows, it is feasible for an agent trajectory to follow $\tau_{\text{targ}(s')}$ from T' to $T^{*\prime}$. Therefore we can construct a feasible MT-TSP-O solution (τ_A^{**}, T^*) by having the agent follow τ_A until time T, follow $\bar{\tau}_A'$ until T', follow $\tau_{\text{targ}(s')}$ from T' to $T^{*\prime}$, and follow τ_A^* from $T^{*\prime}$ to T^*. We push (S', τ_A', T') onto the stack, where τ_A' is the concatenation of τ_A with $\bar{\tau}_A'$, and (τ_A', T') is a prefix of (τ_A^{**}, T^*).

Theorem 3. *If an MT-TSP-O instance is infeasible, MTVG-TSP terminates and reports infeasible in finite time.*

Termination of Alg. 2 follows from its search tree being finite, so the remaining claim to prove is that Alg. 1 terminates when performing LFDT computations before executing Alg. 2. To do so, we use the condition on Line 12 $g_{cand}(v') \leq t^f(s) - T$ to prevent the search from generating nodes that are reached after the end time of s, which bounds the number of steps through $\tilde{G}_{vis}$ away from p the search will ever explore. We then apply this bound within the termination proof of A* [9].

5 Experiments

We ran all experiments on an Intel i9-9820X 3.3GHz CPU with 128 GB RAM. As a baseline, we implemented a sampled-points based method detailed in Section 5.1, based on prior work that solve special cases of the MT-TSP-O [14,19,23]. We test MTVG-TSP and the baseline on 570 problem instances, where an instance consists of a depot location, trajectories and time windows of targets, and an obstacle grid. An example instance is shown in Fig. 3, and we show a solution to an example instance in the video in the supplementary material. For each method, we measured the computation time for the method to obtain its first feasible solution for each instance, setting an upper limit of 300 s.

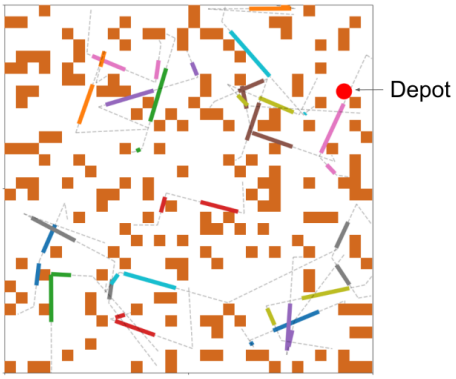


Fig. 3. Example 20-target instance. Each target's time window lengths sum to 26 s.

5.1 Baseline

For our baseline, based on [14, 19, 23], we sample the trajectory of each target within its time windows into points in space-time, such that if we concatenated

all of a target's time windows, the points would be spaced uniformly in time. Next, we plan the shortest obstacle-free path in space between each pair of points using [4], then convert the path into a trajectory where the agent moves at max speed along the path, then waits with zero velocity at the final position until the final time. If the agent cannot reach the final position by the final time, travel between those two points is infeasible. After computing these trajectories, we pose a generalized traveling salesman problem (GTSP) to find a sequence of points to visit. We then formulate the GTSP as an integer linear program (ILP) as in [12], but without subtour elimination constraints. Subtours are only possible if trajectories of two targets intersect exactly in space and time, and we ensure this does not occur. We solve the ILP using Gurobi, obtaining a sequence of points, then concatenate the trajectories between every consecutive pair of points in the sequence to obtain a MT-TSP-O solution.

Since we are not guaranteed to get a feasible solution for a fixed number of sample points, we initialize the algorithm with 10 points per target. If the ILP is infeasible, we increase the number of points by 10 and attempt to solve the ILP again. We repeat this process until the ILP is feasible, then take the first feasible solution Gurobi produces. The computation time reported for an instance is the sum of computation times for all attempted numbers of sample points.

5.2 Generating Problem Instances

We generated two sets of instances, corresponding to Experiment 1 (Section 5.3) and Experiment 2 (Section 5.4). In Experiment 1, we varied the number of targets from 10 to 30 in increments of 10 and varied the sum of each target's time window lengths from 2 s to 50 s in increments of 4, keeping the number of time windows per target fixed to 2. In Experiment 2, we varied the number of targets as in Experiment 1 and the number of time windows per target from 1 to 6, keeping the sum of lengths fixed. We generated 10 instances for each choice of experiment parameters. When varying the sum of a target's time window lengths, we randomly generated the instances with the longest windows first, then randomly shortened the time windows to generate more instances. When varying the number of windows per target, we first generated instances with 1 window with length equal to 22 s, then randomly split this single window into multiple windows. We generated the instances prior to shortening and splitting time windows as follows to ensure all instances were feasible. First, we randomly sampled an occupancy grid with 20% of cells occupied. Then we initialized the agent at a random depot location p_d in free space. Finally we repeated the following steps N_τ times, starting with $i = 1$, $p^0 = p_d$, and $t^0 = 0$:

1. Sample an interception position p^i in free space for target i.
2. Plan a path in space from p^{i-1} to p_i using [4].
3. Move the agent at a speed βv_{max} along the path until the end of the path at p^i, obtaining an arrival time $t^i = t^{i-1} + d^i/(\beta v_{max})$, where d^i is the distance traveled along the path. Here, we use $\beta = 0.99$ to make the instances challenging without making them borderline infeasible.

4. Sample a piecewise linear trajectory for target i such that the number of linear segments equals the specified number of time windows, the trajectory duration is greater than the specified sum of window lengths, and the trajectory arrives at position p^i at time t^i. For each segment of the trajectory, we sample the velocity direction uniformly at random, then select the speed uniformly at random from the range $[\frac{v_{max}}{8}, \frac{v_{max}}{4}]$. This range is from [20], which studies the MT-TSP. After creating this trajectory, we randomly sample a subset of each segment's time interval to create the time windows.

5.3 Experiment 1: Varying Sum of Time Window Lengths

In this experiment, we varied the number of targets and the sum of each target's time window lengths. The results in Fig. 4 show that as we increase the number of targets, we see wider and wider ranges for the sum of window lengths where MTVG-TSP outperforms the sampled-points method in median and maximum computation time. In these critical ranges, the sampled-points method's computation time is large because it needs a large number of points to find a feasible solution, as shown in Fig. 5, leading to a large underlying integer program. In Appendix A in the supplementary material, we show that these peaks occur when there are large intervals in some target's time windows where no feasible MT-TSP-O solution intercepts the target. In these cases, it is difficult to sample a point in one of the *usable intervals* where some feasible solution does intercept the target, since the combined length of these usable intervals is small relative to the combined length of the time windows.

MTVG-TSP's median computation time varies less significantly than the sampled-points method's, though MTVG-TSP's maximum computation time peaks in the same regions as the sampled-points method's. For example, in Fig. 4 (b), consider the 20-target instance with the largest peak in MTVG-TSP's runtime, occuring when the sum of time window lengths equals 14 s. In this instance, we found that if we decreased the sum of lengths to 10 s, MTVG-TSP found a solution intercepting the same sequence of time windows as it did for the 14 s instance, but with 65% less computation time. The reason for this phenomenon is that in the 14 s instance, there are several paths through G_{tw} that do not exist in the 10 s instance, adding branches to Alg. 2's search tree. Profiling showed that in the 14 s instance, Alg. 2 spent 60% of its time exploring these additional branches. The fact that it returned the same sequence of time windows as in the 10 s instance indicates that the additional paths through G_{tw} are useless: they cannot be part of a cycle corresponding to a feasible MT-TSP-O solution, only adding to computation time. On the other hand, we found that increasing the sum of time window lengths from 14 s to 18 s caused MTVG-TSP to find a different time window sequence than in the 14 s instance, and twice as quickly. In this case, increasing time window lengths added useful paths to G_{tw}.

In Table 1, we compare the costs between the methods in instances where the sampled-points method found a solution[7]. The median percent difference in

[7] MTVG-TSP found a solution in all instances.

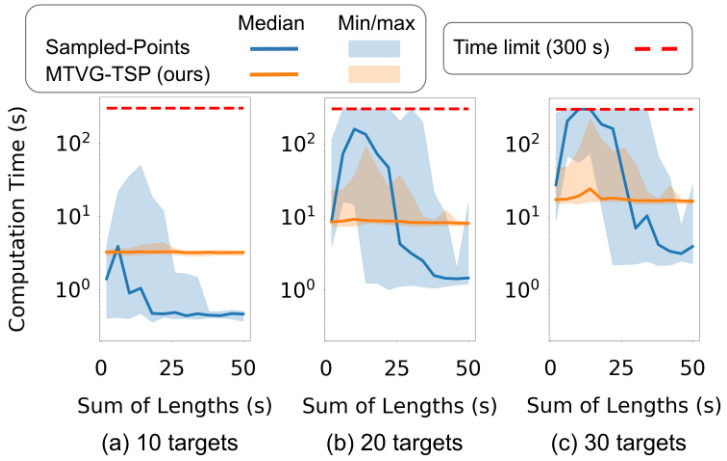


Fig. 4. Time for each algorithm to compute a feasible solution, varying the sum of time window lengths per target while fixing the number of time windows to two. The sampled-points method reached the time limit in 11% of instances without finding a feasible solution. In these cases, the computation time is set equal to the time limit. MTVG-TSP did not reach the time limit in any instance.

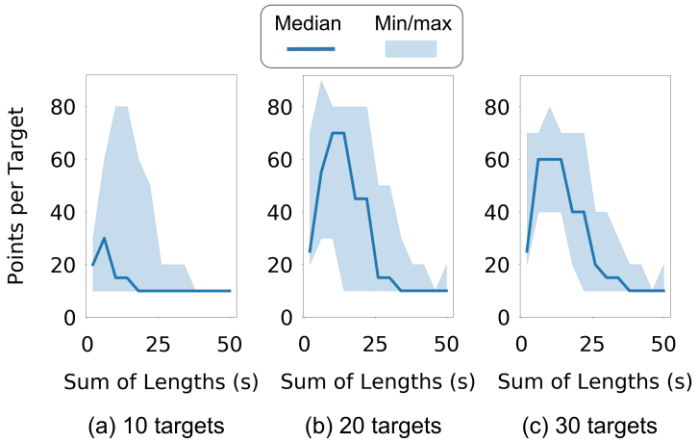


Fig. 5. Maximum attempted number of points per target used by sampled-points method. In instances where the method found a feasible solution, we report the number of points used to obtain the solution. In instances where the method timed out, we report the number of points upon timeout.

Table 1. Comparison of the solution costs T_{MTVG} from our method against the costs T_{PT} from the sampled-points method. Appendix B provides plots of these costs.

	Median	Min	Max
$\frac{T_{MTVG}-T_{PT}}{T_{PT}} * 100\%$	**-0.46%**	**-28%**	**34%**

cost between the methods is small, indicating that both methods often provide similar solution quality. The range of percent differences is large, since we take the first feasible solution from each method as opposed to solving to optimality.

5.4 Experiment 2: Varying Number of Time Windows per Target

In this experiment, we varied the number of targets and number of time windows. Since the sampled-points method does not depend on the number of time windows, only the total set of times covered by the time windows (which we are not varying), we run the sampled-points method for the instances with one window, then show the same runtimes for instances with multiple time windows. As shown in Fig. 6, when we decrease the number of time windows, MTVG-TSP outperforms the sampled-points method in median and max computation time. Both methods' computation times increase with the number of targets. To explain the trends in MTVG-TSP's computation time, we divide its computation time between its three major components: initial visibility computations (Section 3.2), time window graph construction (Section 3.4), and trajectory tree construction (Section 3.5). Fig. 7 shows that computation time for all components increases with the number of targets and number of time windows per target. Increasing either of these quantities increases the total number of time windows, leading to more window-nodes in the time window graph. This requires adding more nodes into the initial visibility graph corresponding to endpoints of targets' trajectories within their time windows, and computing more visible interval sets. Both operations increase initial visibility computation time. More window-nodes also leads to more pairwise LFDT computations when constructing the time window graph. Finally, a larger time window graph leads to a wider and deeper trajectory tree, leading to a larger number of MTVG constructions and searches during trajectory tree construction.

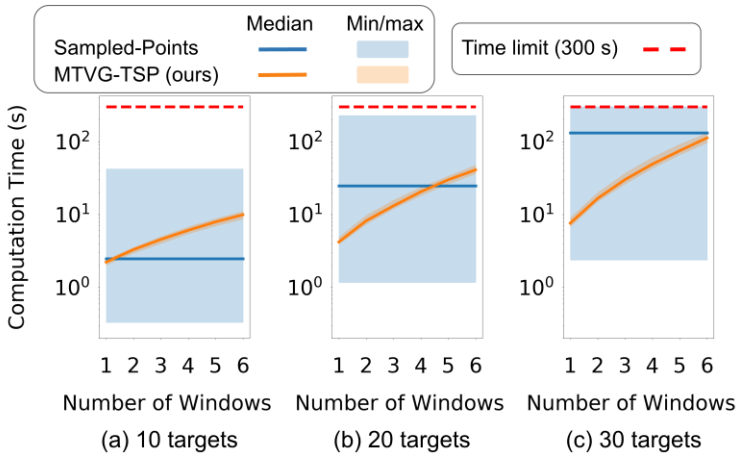


Fig. 6. Time for each algorithm to compute a feasible solution, varying the number of time windows.

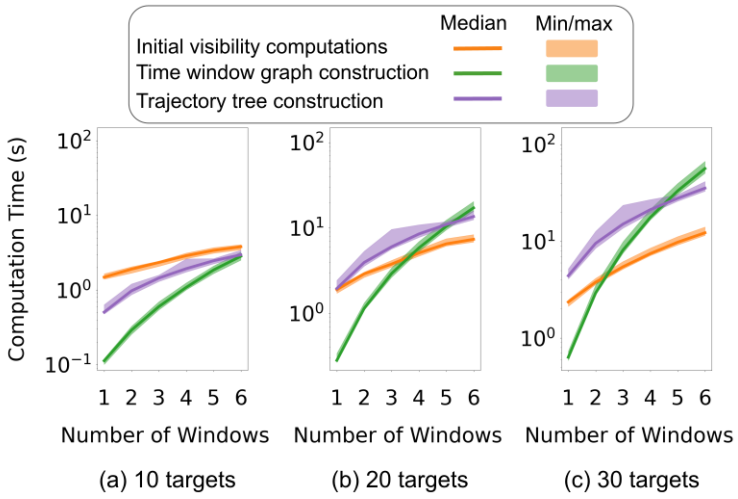


Fig. 7. Breaking down computation time for MTVG-TSP.

We similarly break down the timing for the sampled-points method in this experiment in Appendix B, Fig. 10, dividing time between GTSP graph construction (finding trajectories between all pairs of points) and GTSP solve time. GTSP solve time exceeds graph construction time when we have 20 or more targets.

6 Conclusion

In this paper, we presented MTVG-TSP, a complete algorithm for the moving target traveling salesman problem with obstacles, leveraging a novel graph called a moving target visibility graph (MTVG). We showed that for a range of time window lengths, our algorithm takes less median and maximum time to find feasible solutions than prior methods. Future directions for this work are to incorporate kinodynamic constraints on the agent and involve multiple agents.

Acknowledgments. This material is partially based upon work supported by the National Science Foundation under Grant No. 2120219 and 2120529. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Bourjolly, J.M., Gurtuna, O., Lyngvi, A.: On-orbit servicing: a time-dependent, moving-target traveling salesman problem. International Transactions in Operational Research **13**(5), 461–481 (2006)
2. Choset, H., Lynch, K.M., Hutchinson, S., Kantor, G.A., Burgard, W.: Principles of robot motion: theory, algorithms, and implementations. MIT press (2005)
3. Choubey, N.S.: Moving target travelling salesman problem using genetic algorithm. International Journal of Computer Applications **70**(2) (2013)
4. Cui, M., Harabor, D.D., Grastien, A.: Compromise-free pathfinding on a navigation mesh. In: IJCAI. pp. 496–502 (2017)
5. De Berg, M.: Computational geometry: algorithms and applications. Springer Science & Business Media (2000)
6. Dumas, Y., Desrosiers, J., Gelinas, E., Solomon, M.M.: An optimal algorithm for the traveling salesman problem with time windows. Operations research **43**(2), 367–371 (1995)
7. Englot, B., Sahai, T., Cohen, I.: Efficient tracking and pursuit of moving targets by heuristic solution of the traveling salesman problem. In: 52nd IEEE Conference on Decision and Control. pp. 3433–3438 (2013)
8. Groba, C., Sartal, A., Vázquez, X.H.: Solving the dynamic traveling salesman problem using a genetic algorithm with trajectory prediction: An application to fish aggregating devices. Computers & Operations Research **56**, 22–32 (2015)
9. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics **4**(2), 100–107 (1968)
10. Helvig, C.S., Robins, G., Zelikovsky, A.: The moving-target traveling salesman problem. Journal of Algorithms **49**(1), 153–174 (2003)
11. Jiang, Q., Sarker, R., Abbass, H.: Tracking moving targets and the non-stationary traveling salesman problem. Complexity International **11**(2005), 171–179 (2005)
12. Laporte, G., Mercure, H., Nobert, Y.: Generalized travelling salesman problem through n sets of nodes: the asymmetrical case. Discrete Applied Mathematics **18**(2), 185–197 (1987)
13. LaValle, S.M.: Planning algorithms. Cambridge university press (2006)
14. Li, B., Page, B.R., Hoffman, J., Moridian, B., Mahmoudian, N.: Rendezvous planning for multiple auvs with mobile charging stations in dynamic currents. IEEE Robotics and Automation Letters **4**(2), 1653–1660 (2019)
15. Liu, Y.H., Arimoto, S.: Path planning using a tangent graph for mobile robots among polygonal and curved obstacles: Communication. The International Journal of Robotics Research **11**(4), 376–382 (1992)
16. Lozano-Pérez, T., Wesley, M.A.: An algorithm for planning collision-free paths among polyhedral obstacles. Communications of the ACM **22**(10), 560–570 (1979)
17. Marlow, D., Kilby, P., Mercer, G.: The travelling salesman problem in maritime surveillance–techniques, algorithms and analysis. In: Proceedings of the international congress on modelling and simulation. pp. 684–690. Citeseer (2007)
18. de Moraes, R.S., de Freitas, E.P.: Experimental analysis of heuristic solutions for the moving target traveling salesman problem applied to a moving targets monitoring system. Expert Systems with Applications **136**, 392–409 (2019)
19. Philip, A.G., Ren, Z., Rathinam, S., Choset, H.: C*: A new bounding approach for the moving-target traveling salesman problem (2023)

20. Philip, A.G., Ren, Z., Rathinam, S., Choset, H.: A mixed-integer conic program for the moving-target traveling salesman problem based on a graph of convex sets (2024)
21. Savelsbergh, M.W.: Local search in routing problems with time windows. Annals of Operations research **4**, 285–305 (1985)
22. Stieber, A.: The multiple traveling salesperson problem with moving targets. Ph.D. thesis, BTU Cottbus-Senftenberg (2022)
23. Stieber, A., Fügenschuh, A.: Dealing with time in the multiple traveling salespersons problem with moving targets. Central European Journal of Operations Research **30**(3), 991–1017 (2022)
24. The CGAL Project: CGAL User and Reference Manual. CGAL Editorial Board, 5.6.1 edn. (2024), https://doc.cgal.org/5.6.1/Manual/packages.html
25. Ucar, U., Işleyen, S.K.: A meta-heuristic solution approach for the destruction of moving targets through air operations. International Journal of Industrial Engineering **26**(6) (2019)
26. Wang, Y., Wang, N.: Moving-target travelling salesman problem for a helicopter patrolling suspicious boats in antipiracy escort operations. Expert Systems with Applications **213**, 118986 (2023)